Available online at www.sciencedirect.com

SciVerse ScienceDirect

Procedia Technology 4 (2012) 355 – 359

Procedia Technology

C3IT-2012

Design Pattern Support System: Help Making Decision in the Choice of Appropriate Pattern

Zakaria Moudam[a], Noureddine Chenfour[a]

[a]*Computer Science Department Faculty of Sciences Dhar Mehraz, University Sidi Mohammed Ben Abdellah,Fez, Morocco*

Abstract

Since the ***Gang of Four*** "GoF" has published their book on Design Patterns, this concept has been greatly expanded and becomes a necessity in software development. The Number of new discovered Design Patterns grows rapidly making it difficult for software developers and others to choose between these amounts of patterns for the appropriate one. Hence, a tool that helps making decision in choosing and selecting suitable patterns to solve a specific problem becomes imperative. In this paper we present a support system for making decision in the choice of patterns. We present a Data Management System and initiate a modeling language for Design Patterns aiming to automate query-based search for patterns with respect to their textual part of the description.

Keywords: Design patterns, XML, Keyword.

1. Introduction

Several studies on Design Patterns emerged dealing with discovering, developing and implementing new Design Patterns [10], [4], aiming the detection, the instantiation and the application of Design Patterns in software development [7], [2], or handling the classification, the categorization or the formalization of these Design Patterns[12], [11], [8], [13]; Others automate code generation from Design Patterns [1], [5]. Existing approaches to formalize Design Patterns generally cover only the formal description of the solution structure of Design Patterns. While the structure of a Design Pattern explains how it is applied in software design, it does not explain when to apply a Design Pattern for a given design problem. With the great amount of Design Patterns discovered the need for a good structure becomes more pressing. The main contribution of this paper is a modeling language using XML and XMI technologies aiming the implementation of a Design Patterns Management System (DPMS) for the categorization, the management and the interrogation of an extensible knowledge base, initially concerned with the 23 Design Patterns of the GoF but extensible to cover other patterns and eventually other

doi:10.1016/j.protcy.2012.05.054

formalisms. We implemented a support tool as a Wizard to help searching and selecting Design Patterns classified by their applicability part. Such tool remains important especially with the amount of Design Patterns actually existing.

2. Design Patterns

Design Patterns are a mechanism for expressing object-oriented design experience [9]. A pattern for software architecture describes a particular recurring design problem that arises in specific design contexts and presents a well-proven generic scheme for its solution. The solution scheme is specified by describing its constituent components, their responsibilities and relationships, and the ways in which they collaborate[6]. Part of the idea of Design Patterns is that patterns have a certain literal form. In the GoF book, patterns typically have these major elements: Intent, Motivation, and Applicability, Structure, Participants, Collaborations, Consequences, Implementation, Sample Code, Known Uses and Related Patterns. Part of the benefit of patterns rises from the discipline of having to verbalize these various aspects. We suppose that textual description like intent or applicability can help understanding the purpose of the Design Pattern.

Table 1: structure of the patterns composite and decorator.

Name	Intent	Applicability (When to use)
Composite	* Compose objects into tree structures to represent part-whole hierarchies. *Treat individual objects and compositions of objects uniformly.	* you want to represent part-whole hierarchies of objects. * you want clients to be able to ignore the difference between compositions of objects and individual objects. Clients will treat all objects in the composite structure uniformly.
Decorator	* Attach additional responsibilities to an object dynamically. * Provide a flexible alternative to subclassing for extending functionality.	* to add responsibilities to individual objects dynamically and transparently, that is, without affecting other objects. * for responsibilities that can be withdrawn. * when extension by subclassing is impractical.

We note that the structure of the pattern composite resemble to the pattern decorator (Fig.1). However, their situations when to use (Applicability) are different (TABLE. 1).

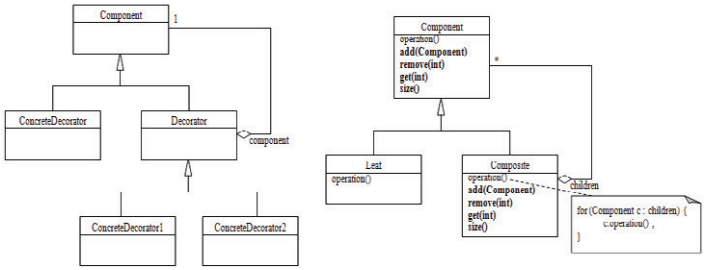

Fig 1: The structure of the patterns composite and decorator.

1. A MODELING LANGUAGE FOR DESIGN PATTERNS

When modeling Pattern we have ensured that the structure is given in such a way that patterns may be easily and effectively understood by others. We propose that patterns are represented by a single XML file divided into two main sections: The first covers the major elements of description of Design Patterns and must contain a set of required components which occur in this order: Intent, Applicability, Consequences, Problem, Solution, aliases, Related Patterns and References and a set of metadata which characterize the pattern. The second deals with the object-oriented side of the pattern.

We focus on the Applicability element which allows us, through a set of criteria we have developed for this reason, the query of the knowledge base not only for documentation purposes but also for assistance as explained in section 5. A fragment of this section for the Chain Of Responsibility pattern looks like:

```
<applicability>
  <critera>
    <statement>more than one object may handle a request, and the handler isn't
          known a priori. The handler should be ascertained</statement>
    <keyword>handle</keyword>
```

XMI.content : The second part presents the object-oriented modeling of the Design Pattern. This section is crucial; it presents the structure of the Design Pattern, the classes and/or objects participating in the Design Pattern and their responsibilities. This allows us -in addition to the abilities offered by XML for query and management- it allows code generation, representation of UML diagrams and inverse engineering. A fragment of the ***XMI.content*** element for the Singleton pattern:

```
<XMI.content>
  <UML:Namespace.ownedElement>
    <UML:Class xmi.id = ' '  name = 'Singleton' visibility = 'public' isSpecification = 'false' isRoot = 'false'
     isLeaf = 'false' isAbstract = 'false' isActive = 'false'>
<UML:Classifier.feature><UML:Operation xmi.id = ''
       name = 'Singleton' visibility = 'private' isSpecification = 'false' ownerScope = 'instance'  isQuery = 'false'
      concurrency = 'sequential'  isRoot = 'false' isLeaf = 'false'  isAbstract = 'false'>
```

2. THE DESIGN PATTERN MANAGEMENT SYSTEM (DPMS)

The solution we propose consists in the implementation of a Design Patterns management system modeled in XML format with respect to the XMI specification. This environment consists of three main layers Fig.4:

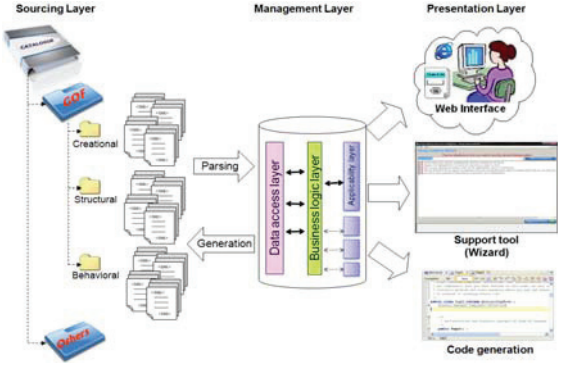

Fig 4: Design Pattern Management System.

Sourcing Layer

This is a storage that contains all possible catalogues (GoF in particular) divided into groups of patterns according to their types (creational, structural, and behavioural). We chose to represent each Design Pattern by a single XML file provided with identifiers that characterize the formalism and the category to which it belongs. We provide each Design Pattern with a set of metadata for bibliographic and documentation purposes. We implemented a framework for automatic storage and retrieval of patterns according to the modeling language discussed above.

Management Layer

It consists of two sub-layers business and persistence responsible for routing all requests users implementing traffic between the base and users. In the business layer we implemented an *Applicability Layer* which filters knowledge according to Applicability criteria. The queries are a bit complicated since it treats a literal domain involving textual searching in data based on keywords [3]. There is a hidden side in this layer: the keywords database dedicated to parameterize the Applicability description of Design Patterns is managed in this level. In fact it consists of an XML database dedicated to store all the related keywords.

Presentation Layer

It allows querying the knowledge base not only for bibliographic purposes but also for editing and updating catalogues, interacting with the base locally through a Java platform for administrators and a web environment for end users. It, mainly, provide support assistance to developers and other functionalities dealing with object-oriented side of the framework like code generation, UML representation or object interchange with third applications. The user interface may be a standalone and/or a web solution. Readers can browse the list of catalogues and the corresponding patterns. Locally, this layer enables administrators to manage patterns through the Framework we developed to this end. Several features are available to handle the object-oriented theory and to produce components that interact with database system: The use of XMI language allows exchanging UML models from one tool to another, we take this opportunity to generate class diagram and thereby generate the source code.

3. THE DESIGN PATTERN WIZARD

As mentioned in the early section, we present a support tool based on the applicability part of the description of Design Patterns. We collect a set of criteria from the description of Design Patterns classified by their applicability. In order to extract knowledge from this base we opted for a categorization into a database of pertinent keywords which characterize the statement criteria Fig.8 (a). This is mainly restricted to the applicability part of Design Patterns and can easily be extended to cover other literal description parts. The set of criteria and the corresponding keywords database must be thinking out enough and enough to cover the most knowledge of Design Patterns. The tool provides the end-user with a non-exhaustive list of keywords, that we have collected, to be used automatically or suggests others and the tool will help him to choose the appropriate Design Patterns.

The wizard is a java platform which enables the search and the selection of suitable Design Patterns regards to the situations in which to use the desired Design Pattern: The first constraint involves the selection of keywords that match the scope of the user interest Fig.8 (b). It's an important phase in which the operation intends the filtering and the refining of the user's ideas in order to reduce the search field and have closely results to the desired ones. With the second constraint, the program will suggest a list of all the situations matching the selected keyword. The user is required to read these criteria and select those that best describe the situations he query for. By checking appropriate statements the user is ready to generate the suitable Design Patterns. Therefore, the user may use all the functionalities available like generate code or class diagrams.

4. CONCLUSION AND PERSPECTIVES

The work presented in this paper enables software developers to find the adequate design pattern(s) for a given design problem. It remains, thus, a measure avoiding the impact of the vast amount of design patterns present in different design pattern catalogues. The keyword-based database can be improved to cover other design patterns beyond the GoF book. In order to enlarge the field of keywords we put forward an ontology based on the applicability description of design patterns which, with the test of the Wizard with other catalogues of design patterns remains for future work.

Fig 8: (a) keywords manager interface, (b) Design Pattern Wizard interface.

References

1. Admodisastro, N. S. Palaniappan (2002). A code generator tool for the gamma Design Patterns. Malaysian Journal of Computer Science 15 (2), 94-101.
2. Albin-Amiot, H. (2003). Idiomes et patterns Java: Application à la synthèse de code et la détection. Ph. D. thesis, _Ecole des Mines de Nantes and Université de Nantes.
3. Bergamaschi, S., E. Domnori, F. Guerra (2010). Keymantic: Semantic keywordbased searching in data integration systems. Proceedings of the VLDB Endowment 3 (2).
4. Borne, I. N. Revault (1999). Comparaison d'outils de mise en œuvre de Design Patterns. Object-oriented Patterns, Vol5, Num2.
5. Budinsky, F., M. Finnie, J. Vlissides, P. Yu (1996). Automatic code generation from Design Patterns. IBM Systems Journal, IBM Corp 35 (2), 151-171.
6. Buschmann, F., R. Meunier, H. Rohnet, P. Sommerlad, M. Stal (1996). Pattern-Oriented Software Architecture, Volume 1, A System of Patterns. New York: Jhon Wiley & Sons, Ltd.
7. Conte, A., M. Fredj, I. Hassine, J. Giraudin, D. Rieu (2002). Agap : an environment to design and apply patterns. In IEEE International Conference on Systems, Man and Cybernetics (IEEE SMC), Hammamet, Tunisia.
8. EDEN, A. (2001). Formal speci_cation of object-oriented design. In International Conference on Multidisciplinary Design in Engineering, CSME-MDE 2001, November 21-22, Montreal, Canada.
9. Gamma, E., R. Helm, R. Johnson, J. Vlissides (2002). Design Patterns: abstraction and reuse of object-oriented design. Software pioneers, Broy, M., Denert, E. (Eds.) LNCS. 1, 701-717 Springer{Verlag New York, Inc., New York, NY, USA.
10. Hannemann, J. G. Kiczales (2002). Design Pattern implementation in java and aspectj. Proceedings of the 17th ACM SIGPLAN conference on Object-oriented programming, systems, languages, and applications, OOP-SLA '02 37, 161-173, New York, NY, USA.
11. Kim, S.-K. C. David (2009). A formalism to describe Design Patterns based on role concepts. Formal Aspects of Computing 21 (5), 397{420.
12. Taibi, T. D. Check Ling Ngo (July-August (2003)). Formal specification of Design Patterns - a balanced approach. Journal of Object Technology 2 (4), 127-140.
13. Zimmer, W. (1994). Relationships between Design Patterns. In Pattern Languages of Program Design. Addison-Wesley.